Reminders

- Homework 3 due Monday, Oct. 25
- Synchronization
- Project 2 parts 1,2,3 due Tuesday, Oct. 26
  - Threads, synchronization

Today:
- Project 2 continued (parts 2,3)
- Synchronization

Project 2 Part 1 Questions

- Anything about writing user threads?
- Recall that:
  - sthread_create doesn’t immediately run the new thread
  - sthread_exit can ignore its ret argument
- Recall how stacks are allocated
  - sthread_new_ctx = creates a new stack and makes it ready to run after first context switch
  - sthread_new_blank_ctx = create new stack but don’t initialize. Suitable to use as old parameter to switch().
- Read .h files for function specs!

Synchronization

Why do we need it?
- Ensure correct and efficient cooperation
- Prevent race conditions

How?
- Protect code in critical sections
  - Allow at most one process/thread in critical section
  - Maintain fairness & progress
  - Don’t make deadlocks

Synchronization Solutions

High-level
- Monitors
- Java synchronized method

OS-level support
- Special variables – mutexes, semaphores, condition vars
- Message passing primitives

Low-level support
- Disable/enable interrupts
- Atomic instructions

+ Software algorithms ...

Disabling/Enabling Interrupts

Thread A:
- disable_interrups()
- critical_section()
- enable_interrups()

Thread B:
- disable_interrups()
- critical_section()
- enable_interrups()

Prevents context-switches during execution of CS

In Linux: cli(), sti()

Sometimes necessary
- E.g. to prevent further interrupts during interrupt handling
- Problems?

Hardware support

Atomic instructions:
- Test and set
- Swap
- Compare-exchange (x86)
- Load-linked store conditional (MIPS, Alpha, PowerPC)

Use these to implement higher-level primitives

E.g. test-and-set on x86:

```
int atomic_test_and_set(int *ptr) {
  int val;
  _asm {
    mov edx, dword ptr [l]; set the pointer to 1
    mov eax, 1 ; load 1 into the accumulator
    cmp edx, 0 ; if edx == 0
    jne next; ; use it

    lock cmpxchgw [edx], eax ; 1 = 1 (and eax = 0)
    jae next; ; else

    mov val, eax ; set eax to be the return val
    mov val, [eax];
  }
  return val;
}
```
Software algorithms

book, p. 193

Example algorithm for two processes 0 and 1, P, does this:
while(1) {
    while(turn != i) ;
    <critical section>
    turn = 1-i;
}

What’s wrong with it?

Hyman’s Algorithm

bool flag[2];
int turn;
void Protocol(int id) {
    while(true) {
        1 flag[id] = true;
        2 while(turn != id) {
            3 while(flag[1-id]) /* Spin */
            4 turn = id;
        }<Critical Section>
        5 flag[id] = false;
        6 <rest of code>
    }
}

P1 executes 1, 2, 3
P0 executes 1, 2, 6
P1 executes 5, 6

Semaphore review

Semaphore = a special variable
- Manipulated atomically via two operations:
  - P (wait)
  - V (signal)
- To access critical section:
  - P(sema)
  - <critical section>
  - V(sema)

Has a counter = number of available resources
Has a queue of waiting threads
- If execute wait() and semaphore is free, continue
- If not, block on that waiting queue
- signal() unblocks a thread if it’s waiting

Synchronization in Project 2

Part 2: write two synchronization primitives
Implement mutex (binary semaphore)
- How is it different from spinlock?
- Need to keep track of lock state
- Need to keep waiting threads on a queue
- In lock(), may need to block current thread
  - Don’t put on ready queue
  - Do run some other thread
- For unlock(), need to take a thread off the waiting queue if available

Condition Variable

A “place” to let threads wait for a certain condition or event to occur while holding a lock (often a monitor lock).

It has:
- Wait queue
- Three functions: wait, signal, and broadcast
  - wait – sleep until condition becomes true.
  - signal – event/condition has occurred. If wait queue nonempty, wake up one thread, o.w. do nothing
    - Do not run the wakeup thread right away
  - broadcast – just like signal, except wake up all threads (not just one).
- In part 2, you implement all of these
Condition Variables 2

More about cond_wait(sthread_cond_t cond, sthread_mutex_t lock):
- Called while holding lock!
- Should do the following atomically:
  - Release the lock (to allow someone else to get in)
  - Add current thread to the waiters for cond
  - Block thread until awoken
- After woken up, a thread should reacquire its lock
  before continuing

How are CVs different from semaphores?
More info: man pthread_cond_wait
- We follow the same spec for wait, signal, bcast

No preemption

You get atomic critical sections for free
However, you should understand what to
do if you had preemption
- Mark critical sections with comments
- Describe appropriate protection that might
  apply (e.g. spinlock).

Monitors: preview

- One thread inside at a time
- Lock + a bunch of condition variables (CVs)
- CVs used to allow other threads to access the
  monitor while one thread waits for an event to occur

Part 3 problem

- N cooks produce burgers & place on stack
- M students grab burgers and eat them
- Provide correct synchronization
  - Check with your threads and pthreads!
  - Print out what happens!
- Sample output (rough draft):
  - cook 2 produces burger #5
  - cook 2 produces burger #6
  - cook 3 produces burger #7
  - student 1 eats burger #8
  - student 2 eats burger #6
  - cook 1 produces burger #8
  - student 1 eats burger #5
  -

Synchronization – Important Points

- Necessary when multiple threads have access to
  same data
- Can't use some primitives in interrupt handlers
  - Why? Which ones?
- Don't forget to release lock, semaphore, etc
  - Check all paths
- Synchronization bugs can be very difficult to find
  - Read your code

Homework questions?

- Sleeping barber problem
- Cigarette-smoker problem
Sample synchronization problem

Late-Night Pizza
- A group of students study for cse451 exam
- Can only study while eating pizza
- Each student thread executes the following:
  - while (1) {
    pick up a piece of pizza;
    study while eating the pizza;
  }
- If student finds pizza is gone, the student goes to sleep until another pizza arrives
- First student to discover pizza is gone phones Pizza Hut and orders a new one.
- Each pizza has 5 slices.

Semaphore solution

```
Student {
  while (diligent) {
    P(pizza);
    P(mutex);
    if (num_slices==0) // took last slice
      V(mutex);
    V(deliver);
    study();
  }
}
```

Late-Night Pizza
- Synchronize student threads and pizza delivery thread
- Avoid deadlock
- When out of pizza, order it exactly once
- No piece of pizza may be consumed by more than one student

Condition Variable Solution

```
int slices=0;
Condition order, deliver;
Lock mutex;
bool first = true;

Student() {
  while(diligent) {
    if (slices > 0) {
      slices--;
    } else {
      if (first) {
        order.signal(mutex);
        first = false;
      }
      deliver.wait(mutex);
    }
    mutex.unlock();
    Study();
  }
}
```

```
DeliveryGuy() {
  while(employed) {
    mutex.lock();
    order.wait(mutex);
    mutex.unlock();
    makePizza();
    slices = 0;
    first=true;
    deliver.broadcast(mutex);
    mutex.unlock();
  }
}
```